AI-Powered Sales Development Representative (SDR) Email Outreach System

Backend-fastapi

Main.py

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from email_generation import generate_email
from email_review import review_email
from email_sender import send_email
from threading import Thread
from email_reply import check_for_replies, email_check_loop
from research import perform_research

app = FastAPI()

class ResearchRequest(BaseModel):
    prospect_name: str
    company_name: str
    additional_info: str

class EmailGenerationRequest(BaseModel):
    prospect_name: str
    company_name: str
    research_report: dict
    product_catalog: str

class EmailReviewRequest(BaseModel):
    email_draft: str
```

```
    sample_emails: str


class EmailSendingRequest(BaseModel):
    sender_email: str
    recipient_email: str
    subject: str
    email_body: str


@app.post("/research")
async def research(request: ResearchRequest):
    try:
        research_report = perform_research(request.prospect_name,
request.company_name)
        if isinstance(research_report, dict):
            return {"research_report": research_report}
        else:
            raise HTTPException(status_code=500, detail="Unexpected format in research
report")
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error performing research: {str(e)}")


# Email generation endpoint
@app.post("/generate_email")
async def generate_personalized_email(request: EmailGenerationRequest):
    try:
        email_draft = generate_email(
            request.prospect_name,
            request.company_name,
            request.research_report,
            request.product_catalog
```

```
        )
        return {"email_draft": email_draft}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error generating email: {str(e)}")

# Email review endpoint
@app.post("/review_email")
async def review_personalized_email(request: EmailReviewRequest):
    try:
        review_result = review_email(request.email_draft, request.sample_emails)
        return review_result
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error reviewing email: {str(e)}")

# Email sending endpoint
@app.post("/send_email")
async def send_personalized_email(request: EmailSendingRequest):
    try:
        smtp_server = "smtp.gmail.com"
        smtp_port = 587
        smtp_user = "salesrepresent124@gmail.com"
        smtp_password = "password"  # Update with the correct credentials

        result = send_email(
            request.sender_email,
            request.recipient_email,
            request.subject,
            request.email_body,
            smtp_server,
```

```
            smtp_port,
            smtp_user,
            smtp_password
        )
        return result
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error sending email: {str(e)}")

def start_email_monitoring():
    print("Starting email monitoring in the background...")
    email_check_loop()

@app.on_event("startup")
async def startup_event():
    thread = Thread(target=start_email_monitoring, daemon=True)
    thread.start()
```

Frontend-Streamlit

```
import streamlit as st
import requests

FASTAPI_BASE_URL = "http://localhost:8000"

if 'research_report' not in st.session_state:
    st.session_state.research_report = None
if 'product_catalog' not in st.session_state:
    st.session_state.product_catalog = None
if 'email_draft' not in st.session_state:
```

```
    st.session_state.email_draft = None
if 'sample_emails' not in st.session_state:
    st.session_state.sample_emails = None
if 'improved_email' not in st.session_state:
    st.session_state.improved_email = None

st.title("SDR_AI")

# Step 1: Generate Research Report
with st.form(key='research_form'):
    st.header("1. Generate Research Report")
    col1, col2 = st.columns(2)
    with col1:
        prospect_name = st.text_input("Prospect Name")
    with col2:
        company_name = st.text_input("Company Name")
    additional_info = st.text_area("Additional Information (optional)")

    submit_button = st.form_submit_button("Generate Research Report")

    if submit_button:
        if not prospect_name or not company_name:
            st.error("Please enter both Prospect Name and Company Name.")
        else:
            try:
                response = requests.post(f"{FASTAPI_BASE_URL}/research", json={
                    "prospect_name": prospect_name,
                    "company_name": company_name,
                    "additional_info": additional_info
```

```
            })
            response.raise_for_status()

            data = response.json()
            research_report = data.get("research_report", {})
            st.write("### CrewAI Research Report")
            if research_report:
                st.write(research_report)
                st.session_state.research_report = research_report
            else:
                st.write("No CrewAI research report found.")
        except requests.RequestException as e:
            st.error(f"Failed to generate research report: {e}")

# Step 2: Upload Product Catalog and Generate Personalized Email
if st.session_state.research_report:
    st.write("Research Report:")
    st.text_area('Research Report', st.session_state.research_report, height=300)

    st.header("2. Upload Product Catalog")
    product_catalog_file = st.file_uploader('Upload Product Catalog (TXT file)', type='txt')

    if product_catalog_file is not None:
        st.session_state.product_catalog = product_catalog_file.read().decode("utf-8")
        st.write('Product Catalog uploaded successfully!')

        if st.button('Generate Personalized Email'):
            with st.spinner('Generating email...'):
                try:
```

```
            email_response = requests.post(f"{FASTAPI_BASE_URL}/generate_email",
json={
                "prospect_name": prospect_name,
                "company_name": company_name,
                "research_report": st.session_state.research_report,
                "product_catalog": st.session_state.product_catalog
            })
            email_response.raise_for_status()
            st.session_state.email_draft = email_response.json().get("email_draft", "")
            st.write("Generated Email Draft:")
            st.text_area('Email Draft', st.session_state.email_draft, height=300)
        except requests.RequestException as e:
            st.error(f"Failed to generate email: {e}")

# Step 3: Upload Sales Email Templates
if st.session_state.email_draft:
    st.header("3. Upload Sales Email Templates")
    sample_emails_file = st.file_uploader('Upload Sales Email Templates (TXT file)',
type='txt')

    if sample_emails_file is not None:
        st.session_state.sample_emails = sample_emails_file.read().decode("utf-8")
        st.write('Sales Email Templates uploaded successfully!')

        if st.button('Review and Improve Email'):
            with st.spinner('Reviewing email...'):
                try:
                    review_response = requests.post(f"{FASTAPI_BASE_URL}/review_email",
json={
                        "email_draft": st.session_state.email_draft,
```

```
                "sample_emails": st.session_state.sample_emails
            })
            review_response.raise_for_status()
            improved_result = review_response.json()
            st.session_state.improved_email = improved_result['improved_email']
            st.write("Reviewed and Improved Email Draft:")
            st.text_area('Improved Email Draft', st.session_state.improved_email,
height=300)
        except requests.RequestException as e:
            st.error(f"Failed to review email: {e}")

# Step 4: Send Email
if st.session_state.improved_email:
    st.header("4. Send Email")
    sender_email = st.text_input('Sender Email')
    recipient_email = st.text_input('Recipient Email')
    subject = st.text_input('Email Subject')

    if st.button('Send Email'):
        if sender_email and recipient_email and subject:
            try:
                email_response = requests.post(f"{FASTAPI_BASE_URL}/send_email", json={
                    "sender_email": sender_email,
                    "recipient_email": recipient_email,
                    "subject": subject,
                    "email_body": st.session_state.improved_email
                })
                email_response.raise_for_status()
                st.success("Email sent successfully!")
            except requests.RequestException as e:
```

```
            st.error(f"Failed to send email: {e}")
        else:
            st.error("Please provide all required fields: Sender Email, Recipient Email, and Subject.")
```

Research.py

```
import requests
from crewai import Agent, Task, Crew
from langchain_ollama import ChatOllama
from typing import Dict, Any

SERPER_API_URL = "https://google.serper.dev/search"
SERPER_API_KEY = "a939292e37152b3d04a7e1539be304105c47eb98"


llm = ChatOllama(
    model="llama3.1",
    base_url="http://localhost:11434"
)

research_agent = Agent(
    role="Research Assistant",
    goal="""Generate a detailed research report about a company prospect by analyzing retrieved search content,
    including recent news, company overview, key contacts, and industry trends.""",
    backstory="You are an AI-powered research assistant skilled at extracting relevant information from search results.",
    allow_delegation=False,
    verbose=True,
    llm=llm,
```

```
    max_tokens=200
)

def serper_search(query: str) -> Dict[str, Any]:
    """
    Search Google using Serper API.
    """
    headers = {
        "X-API-KEY": SERPER_API_KEY
    }
    data = {
        "q": query
    }

    response = requests.post(SERPER_API_URL, json=data, headers=headers)
    print(response.json())
    response.raise_for_status()

    return response.json()

def perform_research(prospect_name: str, company_name: str) -> Dict[str, Any]:
    """
    Perform research using Serper search results and generate a detailed report using CrewAI.
    """
    search_query = f"{prospect_name} {company_name}"
    search_results = serper_search(search_query)
```

```
    task_description = f"Research and summarize information about {company_name} based on search results.Take the input data and generate a detailed research report.Don't generate any fake data."
    task = Task(
        description=task_description,
        agent=research_agent,
        expected_output="A detailed research report based on the search results.Should not be fake only genuine and data from input.",
        input_data=search_results
    )
    crew = Crew(
        agents=[research_agent],
        tasks=[task],
        verbose=True
    )
    crew_result = crew.kickoff()


    result={"research_report": crew_result}


    return result
```

email_generation.py

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_ollama.llms import OllamaLLM

# Define the prompt template
template = """
Create a personalized email for a Sales Development Representative.and i should not add any changes it should be like i am drafting the mail.

Prospect Name: {prospect_name}
```

Company Name: {company_name}

add company name and prospect name in the email draft.

Research Report:

Company Overview: {company_overview}

Recent News: {recent_news}

Industry Trends: {industry_trends}

Product Catalog: {product_catalog}

Email Draft:

Hi {prospect_name},

Best regards,

Agni Prasanth,

Sales Development Representative

"""

model = OllamaLLM(model="llama3.1")

prompt = ChatPromptTemplate.from_template(template)

chain = prompt | model

```
def generate_email(prospect_name, company_name, research_report,
product_catalog):
  formatted_report = {
    "company_overview": research_report.get("company_overview", "No data
available"),
    "key_contacts": research_report.get("key_contacts", "No data available"),
    "recent_news": research_report.get("recent_news", "No data available"),
    "industry_trends": research_report.get("industry_trends", "No data available")
  }

  input_data = {
    "prospect_name": prospect_name,
    "company_name": company_name,
    "company_overview": formatted_report["company_overview"],
    "key_contacts": formatted_report["key_contacts"],
    "recent_news": formatted_report["recent_news"],
    "industry_trends": formatted_report["industry_trends"],
    "product_catalog": product_catalog
  }

  response = chain.invoke(input_data)

  return response.strip()
```

email_review.py

```
from langchain_ollama.llms import OllamaLLM

# Initialize Ollama LLM
model = OllamaLLM(model="llama3.1")
```

```
def review_email(email_draft, sample_emails):
    improve_prompt = f"""
    Improve the following email draft using best practices from the provided sales-winning email templates. Make sure to format the email as if you are drafting it directly. Do not add any extra content or suggestions. The email should not contain any asterisks and must start with "Dear"- prospect name. Do not include any introductory text like "here's your potential response".
    dont add book link or any other links.
    Email Draft:
    {email_draft}

    Sales Winning Email Templates:
    {sample_emails}

    Improved Email Draft:

    """

    improved_email_response = model(improve_prompt)
    improved_email = improved_email_response.strip()




    return {
        "improved_email": improved_email,
    }
```

Email_sender.py

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
```

```
from email.utils import formataddr
from email.mime.application import MIMEApplication

def send_email(sender_email, recipient_email, subject, body, smtp_server, smtp_port, smtp_user, smtp_password):

  try:
    if not body.startswith("Dear"):
      body = body.lstrip().lstrip("Dear").lstrip()
    msg = MIMEMultipart()
    msg['From'] = formataddr(('Analytics Sindhya', sender_email))
    msg['To'] = recipient_email
    msg['Subject'] = subject

    msg.attach(MIMEText(body, 'plain'))

    with smtplib.SMTP(smtp_server, smtp_port) as server:
      server.starttls()
      server.login(smtp_user, smtp_password)
      server.send_message(msg)

    return {"status": "success", "message": "Email sent successfully"}
  except Exception as e:
    return {"status": "error", "message": str(e)}
```

email_reply.py

```
import imapclient
import email
import smtplib
from email.mime.text import MIMEText
```

```
from email.mime.multipart import MIMEMultipart
from langchain_ollama.llms import OllamaLLM
import schedule
import time
from threading import Thread
import json
import os

IMAP_SERVER = 'imap.gmail.com'
IMAP_USER = 'salesrepresent124@gmail.com'
IMAP_PASSWORD = 'password'
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587

# File paths for JSON storage
DB_FOLDER = 'db'
PROSPECTS_FILE = os.path.join(DB_FOLDER, 'prospects.json')
SENT_EMAILS_FILE = os.path.join(DB_FOLDER, 'sent_emails.json')
REPLIED_THREADS_FILE = os.path.join(DB_FOLDER, 'replied_threads.json')

# Initialize LLM model
model = OllamaLLM(model="llama3.1")

# Helper functions for JSON storage
def load_json(file_path):
    if os.path.exists(file_path):
        with open(file_path, 'r') as file:
            return json.load(file)
    return {}
```

```
def save_json(file_path, data):
    with open(file_path, 'w') as file:
        json.dump(data, file, indent=4)

# Create initial empty JSON files if they don't exist
def initialize_json_files():
    if not os.path.exists(PROSPECTS_FILE):
        save_json(PROSPECTS_FILE, {})
    if not os.path.exists(SENT_EMAILS_FILE):
        save_json(SENT_EMAILS_FILE, {})
    if not os.path.exists(REPLIED_THREADS_FILE):
        save_json(REPLIED_THREADS_FILE, {})

def insert_prospect(name, company, additional_info=''):
    prospects = load_json(PROSPECTS_FILE)
    prospects[name] = {"company": company, "additional_info": additional_info}
    save_json(PROSPECTS_FILE, prospects)

def insert_sent_email(to_address, original_subject, response_body, thread_id):
    sent_emails = load_json(SENT_EMAILS_FILE)
    email_entry = {"to_address": to_address, "original_subject": original_subject,
"response_body": response_body, "thread_id": thread_id}
    sent_emails[len(sent_emails) + 1] = email_entry
    save_json(SENT_EMAILS_FILE, sent_emails)

def insert_replied_thread(thread_id):
    replied_threads = load_json(REPLIED_THREADS_FILE)
    replied_threads[thread_id] = True
    save_json(REPLIED_THREADS_FILE, replied_threads)
```

```
def has_replied_thread(thread_id):
    replied_threads = load_json(REPLIED_THREADS_FILE)
    return replied_threads.get(thread_id, False)

def is_reply_to_sent_email(thread_id):
    sent_emails = load_json(SENT_EMAILS_FILE)
    for email_entry in sent_emails.values():
        if email_entry.get("thread_id") == thread_id:
            return True
    return False

# Check for replies in the inbox
def check_for_replies():
    try:
        mail = imapclient.IMAPClient(IMAP_SERVER, ssl=True)
        mail.login(IMAP_USER, IMAP_PASSWORD)
        mail.select_folder('INBOX', readonly=True)

        uids = mail.search(['UNSEEN'])

        if not uids:
            print("No new emails.")
            return

        for uid in uids:
            raw_message = mail.fetch([uid], ['BODY[]', 'FLAGS', 'INTERNALDATE'])
            raw_email = raw_message[uid][b'BODY[]']
            msg = email.message_from_bytes(raw_email)
```

```
            subject = msg.get('Subject', 'No Subject')
            from_address = email.utils.parseaddr(msg.get('From'))[1]
            message_id = msg.get('Message-ID')
            in_reply_to = msg.get('In-Reply-To')
            references = msg.get('References')

            thread_id = in_reply_to or references or message_id or str(uid)

            if has_replied_thread(thread_id) or is_reply_to_sent_email(thread_id):
                print(f"Skipping email from {from_address} as it is a reply to one of our sent emails or already replied.")
                continue  # Skip if already replied to this thread or if it's a reply to one of our emails

            if not in_reply_to and not references:
                print(f"Skipping email from {from_address} as it is not a reply.")
                continue  # Skip if it's not a reply

            body = extract_body(msg)

            response = generate_response(body)

            send_response(from_address, subject, response, thread_id)

            insert_replied_thread(thread_id)

        print("Email check and response sent.")

    except Exception as e:
```

```
        print(f"Error checking emails: {str(e)}")


# Extract body from email
def extract_body(msg):
    if msg.is_multipart():
        for part in msg.walk():
            if part.get_content_type() == 'text/plain':
                return part.get_payload(decode=True).decode(part.get_content_charset(), 'ignore')
    else:
        return msg.get_payload(decode=True).decode(msg.get_content_charset(), 'ignore')


    return "No text part found."


# Generate a response using the model
def generate_response(email_body):
    prompt = f"""
    You are an AI sales assistant. Based on the following email received from a prospect, please draft a professional reply.
    Make sure to format the email as if you are drafting it directly. Do not add any extra content or suggestions. The email should not contain any asterisks and must start with "Dear"- prospect name. Do not include any introductory text like "here's your potential response"


    Email received from the prospect:
    {email_body}


    The response should acknowledge the prospect's queries, provide clear answers, and encourage the prospect to engage further. Use a friendly yet formal tone.
    """
```

```
    response = model(prompt)

    return response.strip()

# Send the response via email
def send_response(to_address, original_subject, response_body, thread_id):
    msg = MIMEMultipart()
    msg['From'] = IMAP_USER
    msg['To'] = to_address
    msg['Subject'] = f"Re: {original_subject}"

    msg.attach(MIMEText(response_body, 'plain'))

    try:
        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
            server.starttls()
            server.login(IMAP_USER, IMAP_PASSWORD)
            server.sendmail(IMAP_USER, to_address, msg.as_string())
            print(f"Response sent to {to_address}")
            insert_sent_email(to_address, original_subject, response_body, thread_id)
    except Exception as e:
        print(f"Error sending email: {str(e)}")

# Email checking loop
def email_check_loop():
    while True:
        schedule.run_pending()
        time.sleep(1)
```

```
schedule.every(60).seconds.do(check_for_replies)

if __name__ == "__main__":
    initialize_json_files()
    email_thread = Thread(target=email_check_loop)
    email_thread.start()

    print("Email checking service started.")
```